#4

0326

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant :	Loren Christensen
Serial No.:	09/842,446
Filed:	April 26, 2001
Title:	HIGH PERFORMANCE RELATIONAL DATABASE MANAGEMENT SYSTEM
Docket No.:	33557

LETTER

Commissioner for Patents
Washington, D.C. 20231

Sir:

Enclosed is a certified copy of Canadian Patent Application No. 2,319,918; the priority of which has been claimed in the above-identified application.

Respectfully submitted,

PEARNE & GORDON LLP

By: John P. Murtaugh
John P. Murtaugh, Reg. No. 34226

526 Superior Avenue East
Suite 1200
Cleveland, Ohio 44114-1484
(216) 579-1700

Date: 7-16-01

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Washington, D.C. 20231 on the date indicated below.

John P. Murtaugh
Name of Attorney for Applicant(s)

July 16, 2001 John P. Murtaugh
Date Signature of Attorney

Offic de la propri´t´ int llectu ll du Canada

Un organisme d'Industrie Canada

Canadian Intellectual Prop rty Offic

An Agency of Industry Canada

Bureau canadien des brevets

Certification

Canadian Patent Office

Certification

La présente atteste que les documents ci-joints, dont la liste figure ci-dessous, sont des copies authentiques des documents déposés au Bureau des brevets.

This is to certify that the documents attached hereto and identified below are true copies of the documents on file in the Patent Office.

Specification and Drawings as originally filed, with Application for Patent Serial No: **2,319,918**, on September 18, 2000, by **LINMOR TECHNOLOGIES INC.** assignee of Loren Christensen, for "High Performance Relational Database Management System ".

Agent certificateur/Certifying Officer

April 27, 2001

Date

Canada

OPIC CIPO

High Performance Relational Database Management System

Inventor: Loren Christensen

The present invention relates to the parallel processing of relational databases within a high speed data network, and more particularly to a system for the high performance management of relational databases.

The problems seen in high capacity management implementations were only manifested recently with the development of highly scalable versions of relational database management solutions.

The effect of high scalability on the volume of managed objects grew rapidly as industry started increasing the granularity of databases. This uncovered still another problem that typically manifested as processing bottlenecks within the network. As one problem was solved it created another that was previously masked.

Networks are now having to managing ever larger number of network objects as true scalability takes hold, and with vendors developing hardware having ever finer granularity of network objects under management, be they via SNMP or other means, the number of objects being monitored by network management systems is now in the millions. Database sizes are growing at a corresponding rate, leading to increased processing times. Finally, the applications that work with the processed data are being called upon to deliver their results in real-time, or near-real-time, thereby adding yet another demand on more efficient database methods.

In typical management implementations, when scalability processing bottlenecks appear in one area, a plan is developed and implemented to eliminate them, at which point they typically will just "move" down the system to manifest themselves in another area. Each subsequent processing bottleneck is uncovered through performance bench marking measurements once the previous hurdle has been cleared.

The present invention is directed to a high performance relational database management system. The system, leveraging the functionality of a high speed communications network, comprises receiving collected data objects from at least one data collection node using at least one performance monitoring server computer whereby a distributed database is created.

The distributed database is then partitioned into data hunks using a histogram routine running on at least one performance monitoring server computer. The data hunks are then imported into at least one delegated database engine instance located on at least one performance monitoring server computer so as to parallel process the data hunks whereby processed data is generated. The processed data is then accessed using at least one performance monitoring client computer to monitor data object performance.

The performance monitor server computers are comprised of at least one central processing unit. At least one database engine instance is located on the performance monitor server computers on a ratio of one engine instance to one central processing unit whereby the total number of engine instances is at least two so as to enable the parallel processing of the distributed database.

At least one database engine instance is used to maintain a versioned master vector table. The versioned master vector table generates a histogram routine used to facilitate the partitioning of the distributed database. The histogram routine comprises dividing the total number of active object identifiers by the desired number of partitions so as to establish the optimum number of objects per partition, generating an *n* point histogram of desired granularity from the active indices, and summing adjacent histogram routine generated values until a target partition size is reached but not exceeded.

The performance monitor server computer comprises an application programming interface compliant with a standard relational database query language.

For data import, reallocation tracking is automatic, since the histogram ranges are always current.

The application programming interface (API) is implemented by means of standard relational database access method thereby permitting legacy or in-place implementations to be compatible.

For a relational database, partitioning of a very large relations can lead to impressive gains in performance. When certain conditions are met, many common database operations can be applied in parallel to subsections of the data set.

The appearance and retirement of entities in tables is tracked by two time-stamp attributes, representing the time the entity became known to the system, and the time it departed, respectively. Versioned entities include monitored objects, collection classes and network management variables.

If a timeline contains an arbitrary interval spanning two instants start and end, an entity can appear or disappear in one of seven possible relative positions. An entity cannot disappear before it becomes known, and it is not permissible for existence to have a zero duration. This means there are 6 possible endings for the first start position, 5 for the second, and so on until the last.

One extra case is required to express an object that both appears and disappears within the subject interval. The final count of the total number of cases is then

$$1+\sum_{n=1}^{6} n$$

There are twenty-two possible entity existence scenarios for any interval with a real duration. Time domain versioning of tables is a salient feature of the design.

A simple and computationally cheap intersection can be used since the domains are equivalent for both selections. Each element of the table need only be processed once, with both conditions applied together.

The serial nature of the existing accessors precludes their application in reporting on large managed networks. While some speed and throughput improvements have been demonstrated by modifying existing reporting scripts to fork multiple concurrent instances of a program, the repeated and concurrent raw access to the flat files imposes a fundamental limitation on this approach.

In the scalability arena, performance degradation becomes apparent when numbers of managed objects reach a few hundreds.

Today's small computers are capable of delivering several tens of millions of operations per second, and continuing increases in power are foreseen. Such computer systems' combined computational power, when interconnected by an appropriate high-speed network, can be applied to solve a variety of computationally intensive applications. Network computing, when coupled with careful application design, can provide supercomputer-level performance. The network-based approach can also be effective in aggregating several similar multiprocessors, resulting in a configuration that might be economically and technically difficult to achieve even with prohibitively expensive supercomputer hardware.

One common paradigm used in distributed-memory parallel computing is data decomposition, or partitioning. This involves dividing the working data set into independent partitions. Identical tasks, running on distinct hardware can then operate on different portions of the data concurrently. Data decomposition is often favored as a first choice by parallel application designers, since the approach minimizes communication and task synchronization overhead during the computational phase. For a relational database, partitioning of a very large relations can lead to impressive gains in performance. When certain conditions are met, many common database operations can be applied in parallel to subsections of the dataset.

If a table D is partitioned into work units D^0, D^1, $\cdots$, D^n, then unary operator f is a candidate for parallelism, if and only if

$$f(D) = f(D_0)\mathrm{U}f(D_1)\mathrm{U}\cdots f(D_n)$$

Similarly, if a second relation O, is decomposed using the same scheme, then certain binary operators can be invoked in parallel, if and only if

$$f(D,O) = f(D_0,O_0)\mathrm{U}f(D_1,O_1)\mathrm{U}\cdots f(D_n,O_n)$$

The unary operators *projection* and *selection*, and binary ops *union*, *intersection* and *set difference* are unconditionally partitionable. Taken together, these operators are members of a class of problems that can collectively be termed "embarrassingly parallel". This could be understood as so inherently parallel that it is embarrassing to attack them serially.

Certain operators are amenable to parallelism conditionally. *Grouping* and *Join* are in this category. Grouping works as long as partitioning is done by the grouping attribute.

Similarly, a join requires that the join attribute also be used for partitioning. That satisfied, Tables do not grow symmetrically as the number of total managed objects increases. The object and variable tables soon dwarf the others as more objects are placed under management. For one million managed objects and a thirty minute transport interval, the incoming data to be processed can be on the order of 154 Megabytes in size. A million element object table will be about 0.25 Gigabytes at it's initial creation. This file will also grow over time, as some objects are retired, and new discoveries appear. Considering the operations required in the production of a performance report, it is possible to design a parallel database scheme that will allow a parallel join of distributed sub-components of the data and object tables, by using the object identifiers as the partitioning attribute. The smaller attribute, class and variable tables need not be partitioned. In order to make them available for binary operators such as joins, they need only be replicated across the separate database engines. This replication is cheap and easy, given the small size of the files in question.

In order to divide the total number on managed objects among the database engines, a histogram must be generated that will divide indices active at the time of a topology update into the required number of work ranges. Dividing the highest active index by the number of sub-partitions is not an option, since there is no guarantee that retired objects will be linearly distributed throughout the partitions. Histogram generation is a three stage process. First the total number of active object identifiers is divided by the desired number of partitions to discover the optimum number of objects per partition. The second operation is to generate an *n* point histogram of arbitrary granularity. This could be understood as so inherently parallel that it is embarrassing to attack them serially from the active indices. Finally, adjacent histogram values are summed until the target partition sizes are reached, but not exceeded. In order to make the current distribution easily available to all interested processes, a versioned master vector table is created on the prime database engine. The topology and data import tasks refer to this table to determine the latest index division information. The table is maintained by the topology import process.

Objects are instantiated in the subservient topological tables by means of a bulk update routine. Most RDBMS's provide a facility for bulk update. This command allows arbitrarily separated formatted data to be opened and read into a table by the server back end directly. A task is provided, which when invoked, opens up the object table file and reads in each entry sequentially. Each new or redistributed object record is massaged into a format acceptable to an update routine, and the result written to one of *n* temporary copy files or relations, based on the object index ranges in the current histogram. Finally, the task opens a command channel to each back end, and issues the update routine. Finally, update commands are issued to set lastseen times for objects that have either by left the system's management sphere, or been locally reallocated to another back end.

The smaller tables are pre-processed in the same way, and are not divided prior to the copy. This ensures that each back end will see these relations identically. In order to distribute the incoming reporting data across the partitioned database engines, a routine is invoked against the most recent flat file data hunk, and its output treated as a streaming data source. The distribution strategy is analogous to that used for the topology data. The data import transforms the routine output into a series of lines suitable for the back end's copy routine. The task compares the object index of each performance record against the ranges in the current histogram, and appends it to the respective copy file. A command channel is opened to each back end, and the copy command given. For data import, reallocation tracking is automatic, since the histogram ranges are always current.

Application programmers will access the distributed database via an API providing C, C++, TCL and PERL bindings. Upon initialization, the library establishes read-only connections to the partitioned database servers, and queries are be executed by broadcasting selection and join criteria to each. Results returned are aggregated, and returned to the application. To minimize memory requirements in large queries, provision is made for returning the results as either an input stream or cache file. This allows applications to process very large data arrays in a flow through manner.

A limited debug and generally access user interface is provided. This takes the form of an interactive user interface, familiar to many database users. The monitor handles the multiple connections, and use a simple query rewrite rule system to ensure that returns match the expected behavior of a non-parallel database. To prevent poorly conceived queries from swamping the system's resources, a build in limit on the maximum number of rows returned is set at monitor startup. Provision is made for increasing the limit during a session.

Network management is a large field that is expanding in both users and technology. On UNIX networks, the network manager of choice is the Simple Network Management Protocol (SNMP). This has gained great acceptance and is now spreading rapidly into the field of PC networks. On the Internet, Java-based SNMP applications are becoming readily available.

SNMP consists of a simply composed set of network communication specifications that cover all the basics of network management in a method that can be configured to exert minimal management traffic on an existing network.

The current trend is towards hundreds of physical devices, which translates to millions of managed objects. A typical example of an object would be a PVC element (VPI/VCI pair on an incoming or outgoing port) on an ATM (Asynchronous Transfer Mode) switch.

The known difficulties relate either to the lack of a relational database engine and query language in the design, or to memory intensive serial processing in the implementation, specifically access speed scalability limitations, inter-operability problems, custom-designed query interfaces that don't provide the flexibility and ease-of-use that a commercial interface would offer.

The limitations imposed by the lack of parallel database processing operations, and other scalability bottlenecks translates to a limit on the number of managed objects that can be reported on, in a timely fashion.

This invention addresses the storage and retrieval of very large numbers of collected network performance data, allowing database operations to be applied in parallel to subsections of the working data set using multiple instances of a database by making parallel the above operations, which were previously executed serially. Complex performance reports consisting of data from millions of managed network objects can now be generated in real time.

This results in impressive gains in scalability for real-time performance management solution. Each component has its own level of scalability.

Each subsequent bottleneck is uncovered through performance benchmarking measurements once the previous hurdle has been cleared. File transfer inefficiencies were resolved through design optimization, the maximum number of managed objects increased from tens of thousands to hundreds of thousands.

As the number of total managed objects increases, the corresponding object and variable data tables increase at a non-linear rate. For example, it was found through one test implementation that one million managed objects with a thirty-minute data sample transport interval can generate incoming performance management data on the order of 154 Megabytes. A million element object table will be about 250 Megabytes at its initial creation. This file will also grow over time, as some objects are retired, and new discoveries appear.

Considering the operations required in the production of a performance report, it is possible to design a parallel database scheme that will allow a parallel join of distributed sub-components of the data and object tables, by using the object identifiers as the partitioning attribute. The steps involve partitioning data and object tables by index and importing partitioned network topology data delegated to multiple instances of the database engine. Invoking an application routine against the most recent flat file performance data hunk direct output to multiple database engines. Application programming interface is via standard relational database access method and user debug and access interface is via standard relational database access method

Scalability limits are advanced. This invention allows for the achievement of an unprecedented level of monitoring influence.

What is claimed is:

1. A high performance relational database management system, leveraging the functionality of a high speed communications network, comprising the steps of:

(i) receiving collected data objects from at least one data collection node using at least one performance monitoring computer whereby a distributed database is created;

(ii) partitioning the distributed database into data hunks using a histogram routine running on at least one performance monitoring server computer;

(iii) importing the data hunks into at least one delegated database engine instance located on at least one performance monitoring server computer so as to parallel process the data hunks whereby processed data is generated; and

(iv) accessing the processed data using at least one performance client computer to monitor data object performance.

2. The system according to claim 1, wherein the performance monitor server computers are comprised of at least one central processing unit.

3. The system according to claim 2, wherein at least one database engine instance is located on the performance monitor server computers on a ratio of one engine instance to one central processing unit whereby the total number of engine instances is at least two so as to enable the parallel processing of the distributed database.

4. The system according to claim 3, wherein at least one database engine instance is used to maintain a versioned master vector table.

5. The system according to claim 4, wherein the versioned master vector table generates a histogram routine used to facilitate the partitioning of the distributed database.

6. The system according to claim 5, wherein the histogram routine comprises the steps of:

(i) dividing the total number of active object identifiers by the desired number of partitions so as to establish the optimum number of objects per partition;

(ii) generating an *n* point histogram of desired granularity from the active indices; and

(iii) summing adjacent histogram routine generated values until a target partition size is reached but not exceeded.

7. The system according to claim 1, wherein the performance monitor server comprises an application programming interface compliant with a standard relational database query language.

8. A high performance relational database management system, leveraging the functionality of a high speed communications network, comprising:

(i) at least one performance monitor server computer connected to the network for receiving network management data objects from at least one data collection node device whereby a distributed database is created;

(ii) a histogram routine running on the performance monitoring server computers for partitioning the distributed database into data hunks;

(iii) at least two database engine instances running on the performance monitoring server computers so as to parallel process the data hunks whereby processed data is generated; and

(iv) at least one performance monitor client computer connected to the network for accessing the processed data whereby data object performance is monitored.

9. The system according to claim 8, wherein the performance monitoring server computers are comprised of at least one central processing unit.

10. The system according to claim 9, wherein at least one database engine instance is located on the performance monitoring server computers on a ratio of one engine instance to one central processing unit whereby the total number of engine instances for the system is at least two so as to enable the parallel processing of the distributed database.

11. The system according to claim 10, wherein at least one database engine instance is used to maintain a versioned master vector table.

12. The system according to claim 11, wherein the versioned master vector table generates a histogram routine used to facilitate the partitioning of the distributed database.

13. The system according to claim 12, wherein the histogram routine comprises the steps of:

(i) dividing the total number of active object identifiers by the desired number of partitions so as to establish the optimum number of objects per partition;

(ii) generating an *n* point histogram of desired granularity from the active indices; and

(iii) summing adjacent histogram routine generated values until a target partition size is reached but not exceeded.

14. The system according to claim 8, wherein the performance monitor server comprises an application programming interface compliant with a standard relational database query language.

15. The system according to claim 8, wherein at least one performance monitor client computer is connected to the network so as to communicate remotely with the performance monitor server computers.

16. A storage medium readable by an install server computer in a high performance relational database management system including the install server, leveraging the functionality of a high speed communications network, the storage medium encoding a computer process comprising:

(i) a processing portion for receiving collected data objects from at least one data collection node using at least one performance monitoring computer whereby a distributed database is created;

(ii) a processing portion for partitioning the distributed database into data hunks using a histogram routine running on at least one performance monitoring server computer;

(iii) a processing portion for importing the data hunks into at least one delegated database engine instance located on at least one performance monitoring server computer so as to parallel process the data hunks whereby processed data is generated; and

(iv) a processing portion for accessing the processed data using at least one performance client computer to monitor data object performance.

17. The system according to claim 16, wherein the data processing server computers are comprised of at least one central processing unit.

18. The system according to claim 17, wherein at least one database engine instance is located on the data processor server computers on a ratio of one engine instance to one central processing unit whereby the total number of engine instances is at least two so as to enable the parallel processing of the distributed database.

19. The system according to claim 18, wherein one of the database engine instances is designated as a prime database engine instance used to maintain a versioned master vector table.

20. The system according to claim 19, wherein the versioned master vector table generates a histogram routine used to facilitate the partitioning of the distributed database.

21. The system according to claim 20, wherein the histogram routine comprises the steps of:

(i) dividing the total number of active object identifiers by the desired number of partitions so as to establish the optimum number of objects per partition;

(ii) generating an *n* point histogram of desired granularity from the active indices; and

(iii) summing adjacent histogram routine generated values until a target partition size is reached but not exceeded.

22. The system according to claim 16, wherein the performance monitor server comprises an application programming interface compliant with a standard relational database query language.

Figure 1

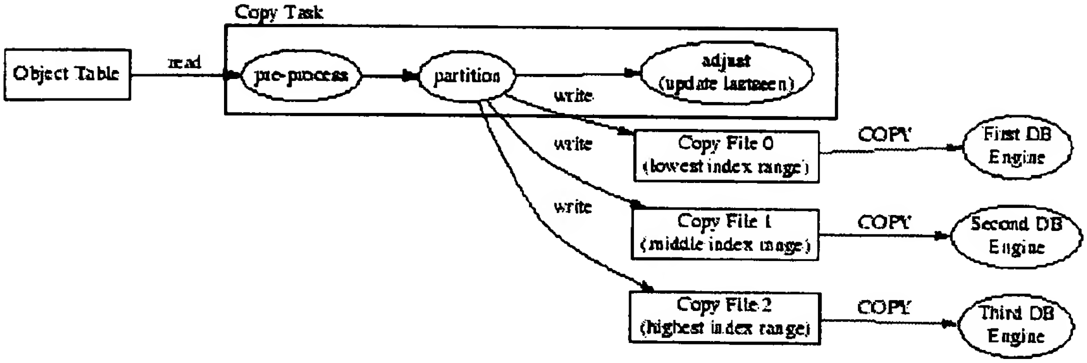


Figure 2

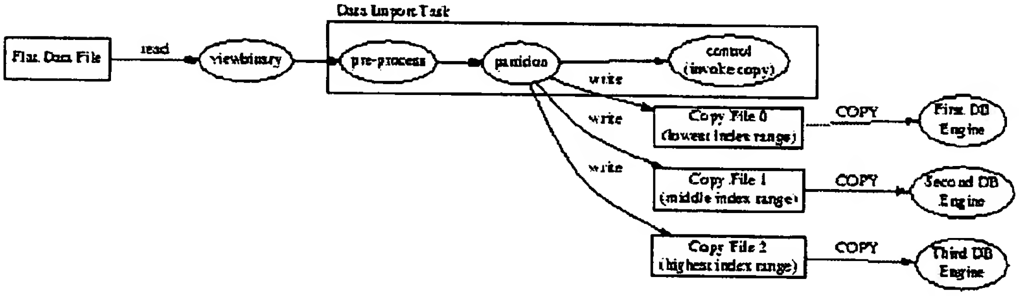


Figure 3

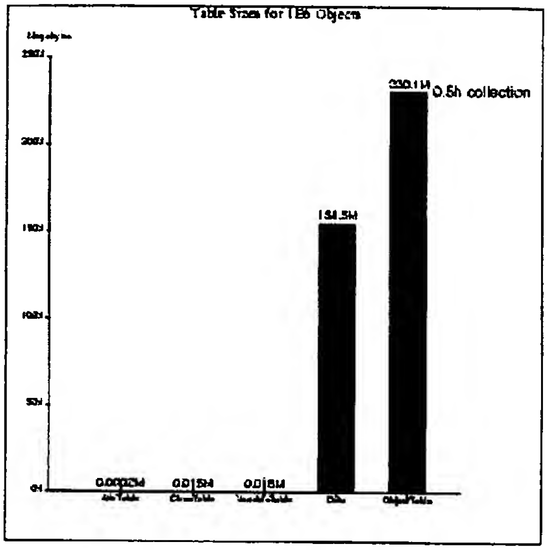
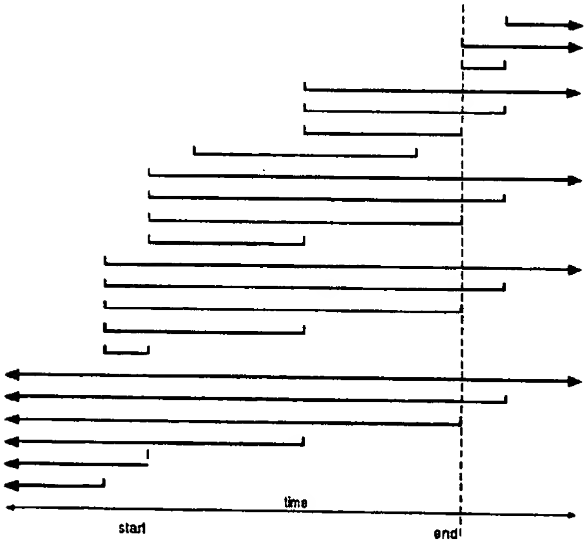


Figure 4


time
start
end